



**ORACLE®**

## 更に進化したストレージ仮想化技術: ZFS

日本オラクル株式会社  
システム事業統括 ソリューション統括本部  
シニアセールスコンサルタント  
野崎 宏明

以下の事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。以下の事項は、マテリアルやコード、機能を提供することをコミットメント（確約）するものではないため、購買決定を行う際の判断材料になさらないで下さい。オラクル製品に関して記載されている機能の開発、リリースおよび時期については、弊社の裁量により決定されます。

OracleとJavaは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。

# スピーカー自己紹介

- 学生時代
  - JDK 1.1でPrologの推論機構を実装し検索エンジンの真似ごと
- 1998年4月(日本)サン・マイクロシステムズ入社
  - ふと気づけば社会人13年目
- ほぼずっとSE (Sはシステム?セールス?)
  - 主にSolaris、開発環境、Sun Rayシンクライアント
  - 今は主に通信事業者様担当
- 1年だけサポートエンジニア
  - Java VMの解析
- Solarisエバンジェリスト(宣教師?!)
  - 最近は特にZFSの宣伝係としての出番増



# 内容

- ZFS 概要
- Oracle Solaris 11 Express 2010.11におけるZFS の主な新機能、拡張点
  - 暗号化
  - 重複排除
  - 性能向上
  - ルートプールに関する拡張点
  - プールの修復に関する拡張点
  - zfs send/recvの拡張点
  - その他の新機能、拡張点



# ZFS 概要



# ZFS: 革新的ファイルシステム

## 拡張性

- 事実上無制限
  - プール容量
  - ファイルシステム容量
  - ファイル数
  - ディレクトリ数
- 圧縮
- 重複排除 **new!**

## 管理性

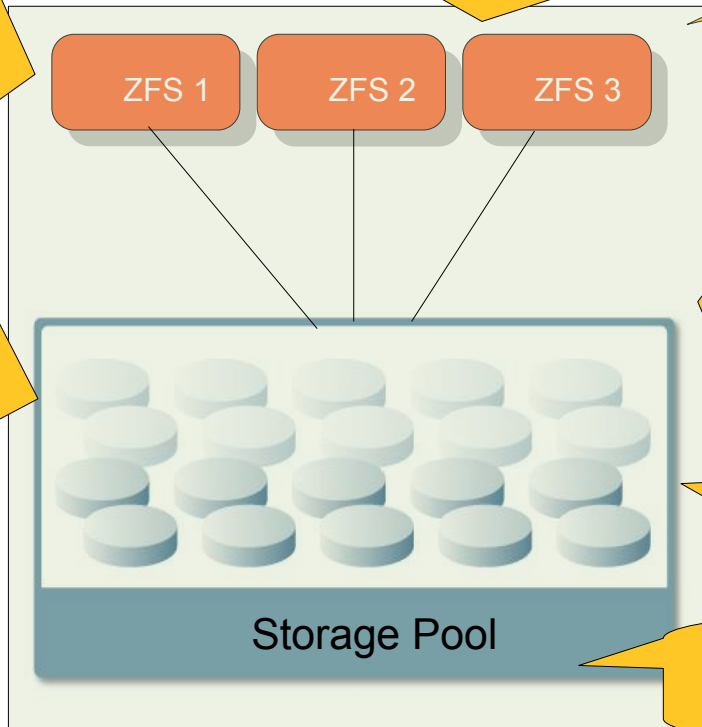
- 極めて簡素
  - プール構造
  - ボリューム管理の統合
  - 属性値による管理

## 機能

- スナップショット
- リモートコピー
- バックアップ
- ファイル共有

## 堅牢性

- Copy-on-Write
- トランザクション処理
- End-to-End チェックサム
- 自己修復
- 暗号化 **new!**



## 性能

- Hybridストレージプール
- Read-modify-writeの排除
- 入出力のパイプライン処理

追加ライセンス費無  
オープンソース

Solaris 11 Expressの  
ルートファイルシステム

# ZFSの歴史と機能比較



日付	build	ver	S11E	ZFSSA	S10	内容
2005/10/31	-	-	-	-	-	オープンソースで公開
2006/03	36	1	-	-	-	Solaris 10 6/06 で製品として初リリース
2006/06	42	3	○	○	○	ダブルパリティ RAID-Z
2007/07	62	6	○	○	○	ブート対応
2007/07	68	7	○	○	○	分離ログデバイス (slog)
2007/11	78	10	○	○	○	2次読込キャッシュ (L2ARC)
2008/05	90	-	○	○	×	COMSTAR 対応
2008/11	-	-	-	-	-	Storage 7000 シリーズ (現ZFSSAシリーズ)
2009/05	114	15	○	○	○	ユーザー・グループ単位容量制限
2009/07	120	17	○	○	○	トリプルパリティ RAID-Z
2009/11	128	21	○	○	×	重複排除
2010/09	149	30	○	未	×	暗号化

build: Nevada build, ver: zpool version, S11E: Solaris 11 Express 2010.11 (Nevada b151a / zpool ver 31), ZFSSA: ZFS Storage Appliance 2010.Q3, S10: Solaris 10 9/10 (zpool ver 21 but dedup not supported)

# Sun ZFS ストレージアプライアンス(ZFSSA)

- Unified Storage / Sun ZFS Storage 7000

- Solaris ベースの NAS ヘッド
- オープンアーキテクチャ
  - データ形式、プロトコル (NFS, FTP, WebDAV, iSCSI, CIFS, FC, IB, NDMP, ..)
  - オープンソースソフトウェア採用
  - 追加ライセンス費用**無**

- ZFS と Flash (SSD)

- Hybridストレージプール

- DTrace Analytics

- リアルタイム



7120

~120TB

7320

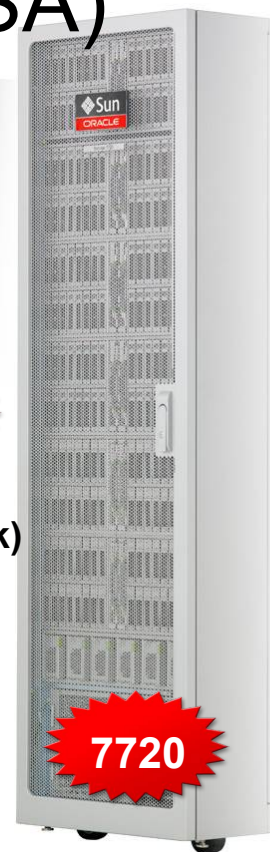
~192TB



7420

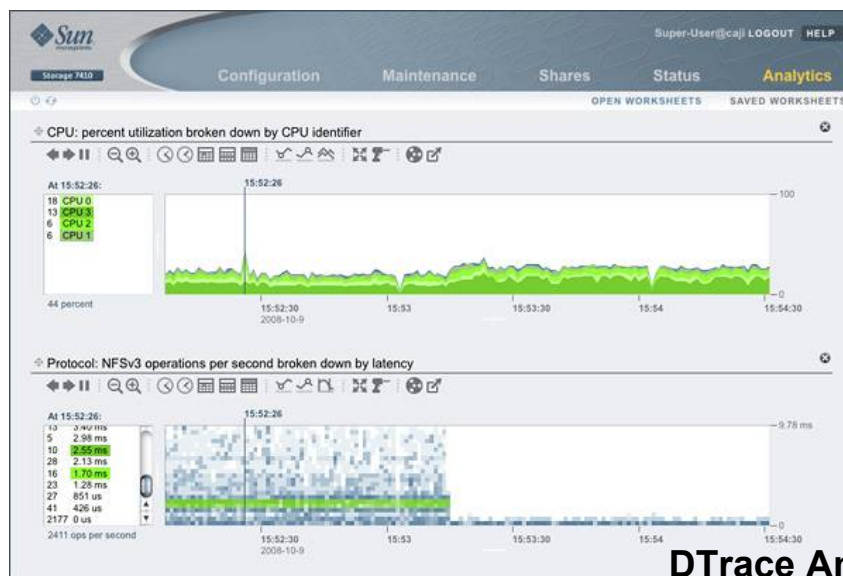
~1152TB

(~480TB/rack)



7720

~720TB/rack



DTrace Analytics

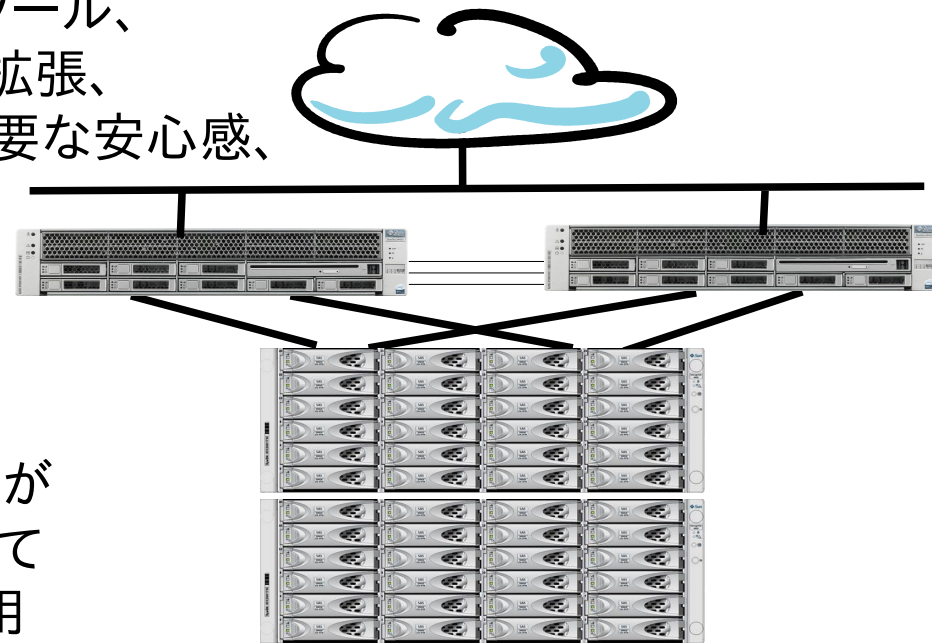


システム性能確認  
容量プランニング  
課題特定

# ZFS 活用事例:

## インターネットイニシアチブジャパン (IIJ) 様

- Storage 7410 によるクラウド用ストレージサービス
- 採用ポイント
  - 各顧客に独立した容量をスムーズにシンプロビジョニング可能
  - 保守性、信頼性、直感的な管理ツール、Flash による比較的安価な性能拡張、全機能標準搭載で追加コスト不要な安心感、手離れの良さといったトータルな価格対容量比
  - 既存システムとの統合(作り込みと自動化)の容易性
  - 「クラウドの向こう側」のユーザーがストレージにかかる負荷を分析して性能や容量のプランニングに活用

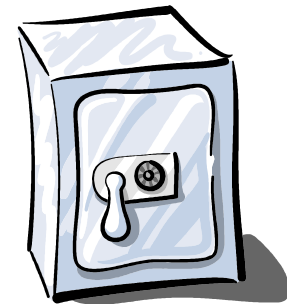





# Oracle Solaris 11 Express 2010.11 ZFSの主な新機能、拡張点

# 暗号化 ZFS

# 暗号化ZFS




- データの内容秘匿のため暗号化鍵でコード化
  - データ暗号化鍵はラップ用鍵で暗号化される
- **これまでどおりの ZFS 管理モデルに統合** 
- 既存プールでも利用可能
  - プールバージョンを30以上にアップグレードする必要あり
- 鍵管理の権限委譲が可能
  - 管理者、エンドユーザー
  - ゾーン
  - 「鍵使用」と「鍵変更」の分掌
    - 外部預託の仕組みも作ることができる
- ZFS I/O 性能への影響 (負荷にも大きく依存)
  - 大まかにランダムI/Oで約7%、シーケンシャルI/Oで約3%を見込む

# 暗号化をどこに設定するか




- ZFS の**データセット単位**で暗号化ポリシーを設定
  - ほとんどのシステムではプールは1-2個だがデータセットは多数
  - 現在 AES-128 / 192 / 256 をサポート
    - デフォルトでは aes-128-ccm
    - 他の方式もサポート可能なデザイン
  - iSCSI、FCoEターゲットは ZVOL を暗号化
    - イニシエータ側での鍵管理無し
  - NFSv2,v3,v4、CIFS (SMB) でデータセット共有可能
    - NAS クライアント側では鍵管理無し

# 暗号化をいつ設定するか

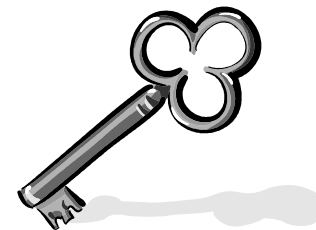
- ・ 暗号化属性値はデータセット**作成時**に指定 
  - ・ copy-on-writeで古いクリアテキストが残るのを避けるため


```
# zfs create -o encryption=on tank/home/darren
Enter passphrase for 'tank/home/darren': xxxxxxxx (←8文字以上)
Enter again: xxxxxxxx
```

- ・ チェックサムは SHA256 になる
- ・ 既存データセットの暗号化ポリシーは**変更できない** 
  - ・ 暗号化アルゴリズムやチェックサムを後から無効化や変更不能

```
# zfs set encryption=off tank/home/darren-new
cannot set property for 'tank/home/darren-new': 'encryption'
is readonly
```

## ラップ用鍵



- **keysource**属性値
  - ユーザー・管理者が管理するラップ用鍵の形式や場所を指定
  - **passphrase**: PKCS#5パスワードベース暗号化を使用
  - **raw / hex**: バイト形式または 16 進テキスト文字列
- ラップ用鍵は派生データセットに**継承される** 
  - クローン時に異なるラップ用鍵を指定可能
    - 新たな暗号化鍵を使用することも可能
      - デフォルトではoriginデータセットの鍵を使用

```
# zfs clone tank/home/darren@now tank/home/darren-new
Enter passphrase for 'tank/home/darren-new': yyyyyyyy (←非表示)
Enter again: yyyyyyyy (←非表示)
```

- ファイルからのラップ用鍵作成例

```
# pktool genkey keystore=file outkey=/rmdisk/stick/mykey \
  keytype=aes keylen=256
# zfs create encryption=aes-256-ccm \
  -o keysource=raw,file:///rmdisk/stick/mykey tank/home/bob
```

# 暗号化ZFSデータセットの作成例



```
# zfs create -o encryption=on build/fs
Enter passphrase for 'build/fs': xxxxxxxx (← 8文字以上、非表示)
Enter again: xxxxxxxx (← 非表示)
# zfs create build/fs/sub
# zfs get -r encryption,keysourc,keystatus,checksum build/fs
NAME                PROPERTY           VALUE              SOURCE
build/fs            encryption         on                 local
build/fs            keysourc           passphrase,prompt  local
build/fs            keystatus          available          -
build/fs            checksum           sha256-mac        local
build/fs/sub        encryption         on                 inherited from build/fs
build/fs/sub        keysourc           passphrase,prompt  inherited from build/fs
build/fs/sub        keystatus          available          -
build/fs/sub        checksum           sha256-mac        inherited from build/fs
```

# データセット暗号化鍵の変更



- ラップ用鍵の変更
  - 既存データは再暗号化**されない** !
  - ユーザーや管理者が管理可能
- 鍵の再生成
  - 以降用いる新しいデータ暗号化鍵を追加
  - NIST 800-57 Recommendations on Key Management
- 鍵変更や再生成は**オンラインで可能** !
  - データセットはマウントされている必要あり
    - 最小限でも鍵が使用可能な状態にする必要あり
  - 鍵変更や再生成の最中もデータセットはマウント・共有されたまま

# ラップ用鍵の変更と鍵の再生成



```
# zfs key -c build/fs
```

```
Enter new passphrase for 'build/fs': zzzzzzzz (← 非表示)
```

```
Enter again: zzzzzzzz (← 非表示)
```

```
# zfs key -K build/fs
```

```
# zfs get creation,rekeydate build/fs
```

NAME	PROPERTY	VALUE	SOURCE
build/fs	creation	火 10月 12 11:12 2010	-
build/fs	rekeydate	火 10月 12 11:17 2010	local

クローンの際にラップ用鍵を変更し、データ暗号化鍵も再生成する例

```
# zfs clone -K -o keysource=raw,file:///rmdisk/stick2/key2 \
  build/fs@snap1 build/sn1fs
```


# ディスク上で何が暗号化されるか

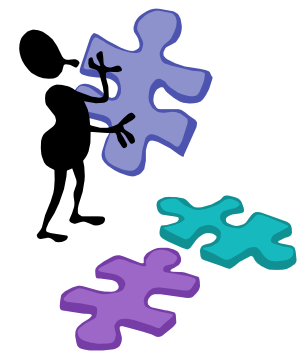
- 暗号化される
  - 全アプリケーションデータ
  - POSIX 層のデータ
  - パーミッション、ACL、所有者
  - ディレクトリ構造
  - 全 ZVOL データ
  - (スナップショットでも同様)
  - (クローンでも同様)
- 暗号化されない
  - プールのメタデータ
  - プールの履歴
  - RAID構成情報
  - データセット属性値
  - データセット名
  - データセットのユーザー属性値

0101011001  
0010101011  
1011010101  
0110011101  
1010110010


データデータデータ  
情報情報情報情報  
データデータデータ  
情報情報情報情報  
データデータデータ  
情報情報情報情報

# 配備にあたっての考慮点

- 暗号化データセットからは**ブートできない** 
  - /var/tmp を別のファイルシステムにすることはできる
  - /tmp は swap がバックエンドなので暗号化できる
    - swap の暗号化方法は付録参照
- encryption=off の zfs sendをencryption=onでrecv不能
  - 逆は可能
    - send 側: encryption=on → recv 側: encryption=off
- 今後の拡張点
  - ラップ用鍵を http/https URI から取得
  - ラップ用鍵を PKCS#11 キーストアから取得
    - Oracle 鍵管理システムとの統合など




# 復号化されたデータについて

- メモリ内キャッシュ (ARC) は多くの復号済データを含む
  - ただし見るためには /dev/kmem へのアクセス特権が必要
- データセットごとの**キャッシュデータ制限**を検討
  - primarycache (メモリ): none, metadata, all 
  - secondarycache (Flash等): none, metadata, all

# Flash デバイスと ZFS 暗号化



- ZFS は 2 つの目的で Flash を有効利用
  - ZIL: ZFS Intent Log
    - 書込が高速なデバイスが求められる
  - L2ARC: メモリとディスクの間としての読込キャッシュ
    - 読込が高速なデバイスが求められる
- データセットを暗号化する際
  - Flash の暗号化も**強く推奨** 
  - 電源断でもデータが保持されるので
  - ZIL は暗号化される (Flash での slog でも同様)
  - L2ARC は暗号化できないので今は無効化で対処

# 重複排除 (deduplication)

# 重複排除 (deduplication)

## ・ **ブロック単位** の重複検知・排除機構

- ! 同一内容データブロックは全体で1コピーだけを格納
- 格納されているブロックと同一内容のブロックはオンライン(オンザフライ)で除去される(格納されない)
- 共通部分は複数ファイルで共有
- メタデータは重複排除対象外
- 重複判定を実行するための専用ディスク領域等は不要

## ・ **チェックサム(SHA256)を使用**して重複判定

- ! 圧縮、暗号化と**同時利用可能** !

## ・ 特に効果の高い適用範囲

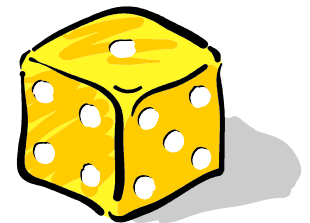
- ! **仮想化** !

- ! **サーバーバックアップ** !

- ! **開発環境** !

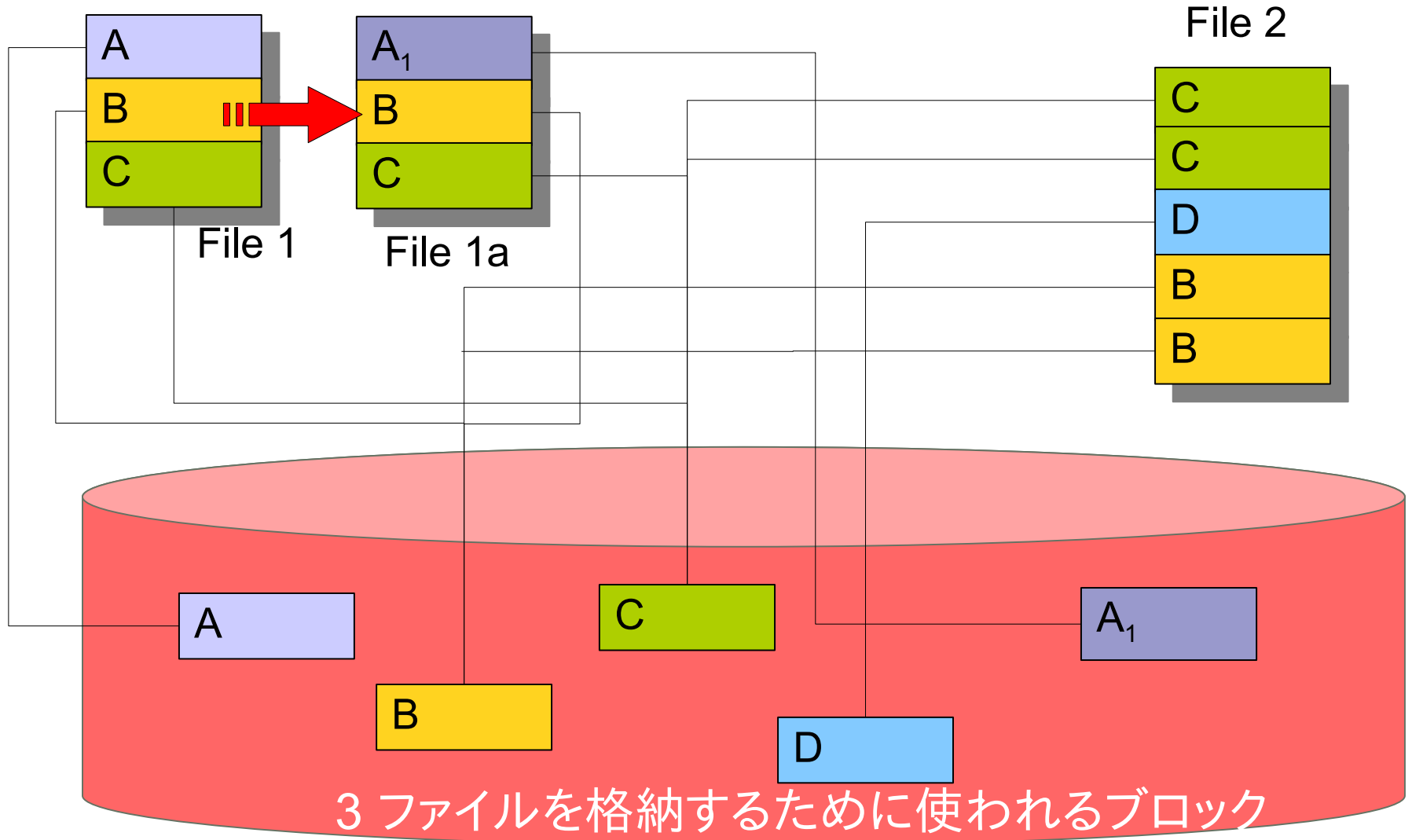
## ・ ZFSに統合済み (**アドオンではない**)

- 特別なハードウェア不要
- デザイン上容量制限無し
- 直近では性能向上を重視



# 重複排除

## ブロックレベルでの重複排除機構



## 重複排除

# 構成方法



- **dedup**属性値をデータセット(スナップショット除く)単位で指定
  - デフォルトではオフ

```
# zfs set dedup=on tank/home
```

- スコープはプール単位
  - dedup=onの全データセットで共通のブロックを共有
  - 重複排除率確認方法 (プール内で dedup 適用データセットのみ計上)

```
# zpool list tank
```

```
NAME SIZE  ALLOC  FREE   CAP    DEDUP  HEALTH  ALTROOT
tank 136G   55.2G  80.8G  40%    2.30x  ONLINE  -
```

- 有効時はzfs listの容量表示に注意
  - 参照サイズの合計が表示される
    - 重複排除分は考慮されず実際のディスク消費量より多く表示される

## 重複排除

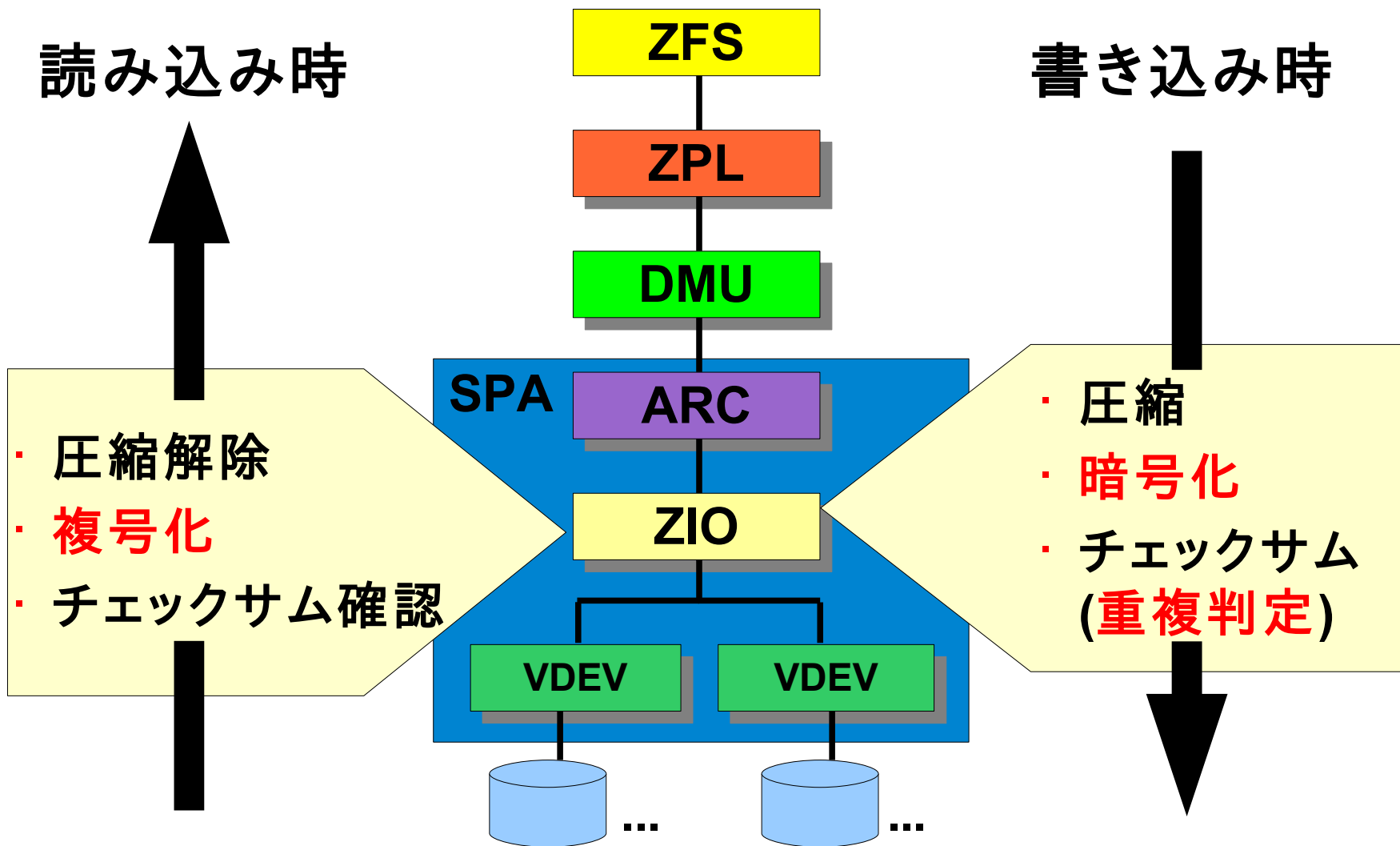
# 重複排除の注意点



- データセット属性値で有効にした**後の書込をオンライン(オンザフライ)**で重複排除
  - 有効にする前に書き込まれたデータは重複排除**されない**
- 重複判定にCPU、ARC (あふれたらL2ARC、次いで**HDD**) を消費
  - # zdb -S <zpool> (事前見積) (zdb -DD <zpool> で事後確認)
    - 重複排除の有効性: dedup ratio > 1.0 (容量節約率の計算はプール単位)
    - 重複排除テーブル (DDT) のサイズ: 約 250 bytes x allocated block数
      - 大まかな見積り:  
recordsize=128KB、重複無しの20TBデータ → DDT 32GB  
recordsize=8KB、重複無しの1TBデータ → DDT 32GB
      - ARCをあふれる場合はL2ARCが非常に効く(典型的なランダムリード)
- チェックサムによるブロック単位での重複判定 (SHA256)
  - コリジョン発生確率  $2^{-256} \doteq 10^{-77}$ 、verifyはさらにbyte単位で比較
- cloneとの組み合わせも検討
  - 重複ブロックが多いことが確実でも丸ごとcloneして差分管理する方が有効な場面
    - 仮想マシンイメージファイルは**clone**、パッチ適用時は**dedup**、等
- 圧縮や暗号化との同時使用も可能
  - 書込時: 圧縮→暗号化→チェックサム(重複排除)

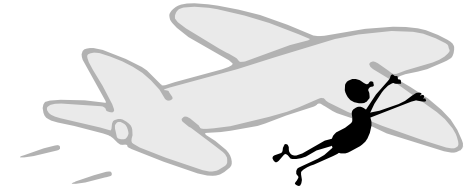


# ZFS 圧縮、暗号化、チェックサム



# ZFSの性能向上

# ZFS の性能向上



- Hybrid ストレージプール
- duty-cycleスケジューラ
- logbias 属性値
- sync 属性値
- RAID-Z/ミラー ハイブリッドアロケーション
- カーネルモード iSCSI ターゲット (COMSTARによる)
- その他の性能向上 (本日は詳細は割愛させていただきます)
  - ブロックアロケータの効率化、高速化
  - scrub/resilver時の先読み
  - raw scrub/resilver
  - ゼロコピー I/O
  - スリムZIL

# Hybrid ストレージプール



- ・ **同期書込** 高速化のためログデバイスを分離(slog)



- ・ エンタープライズ向け SLC Flash
  - ・ NVRAMよりは安価
  - ・ SASファブリックによりクラスタ化も容易

- ・ **ランダム読込** 高速化のためのキャッシュデバイス

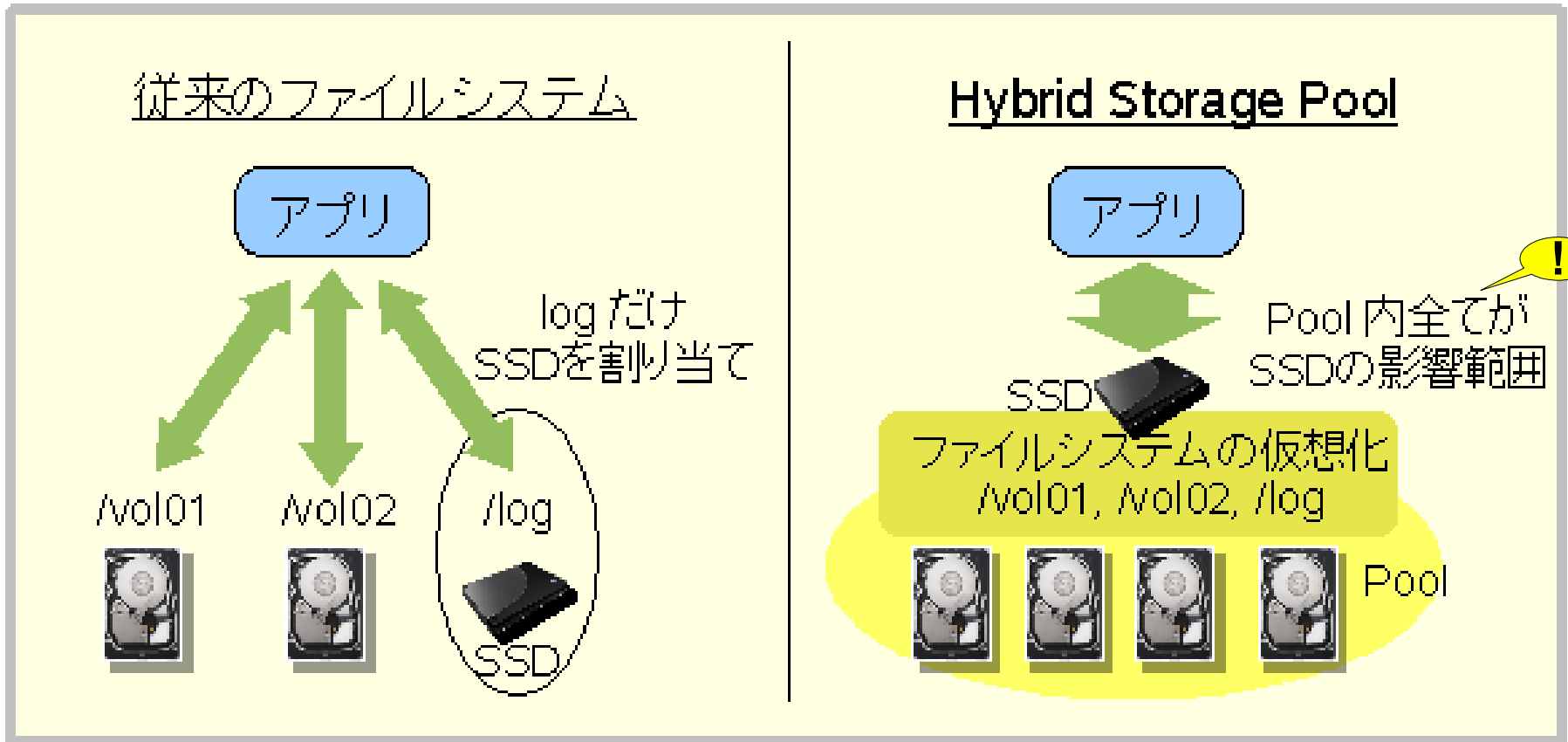


- ・ 安価なコンシューマ向け MLC Flash
  - ・ DRAM からあふれた内容をキャッシュ
    - ・ 非常に大きなサイズのワーキングセットをカバー
  - ・ 単なるキャッシュ
    - ・ 故障によるデータロス、クラスタ化など考慮不要
  - ・ 全てチェックサムがとられている
    - ・ silentエラーのリスク無し



- ・ **低消費電力、大容量のディスク**を1次ストレージにできる

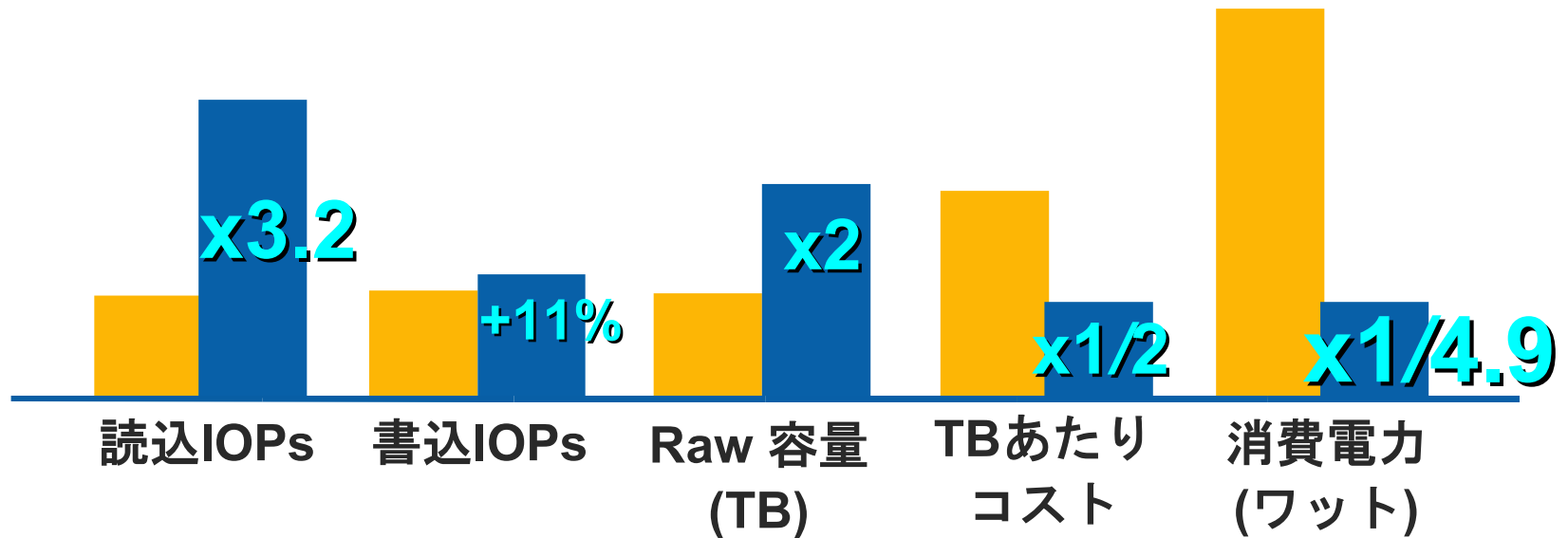
# Hybridストレージプール: 容易な管理



# Hybridストレージプールの性能効果(例)



## Flash無し vs. Flash有り

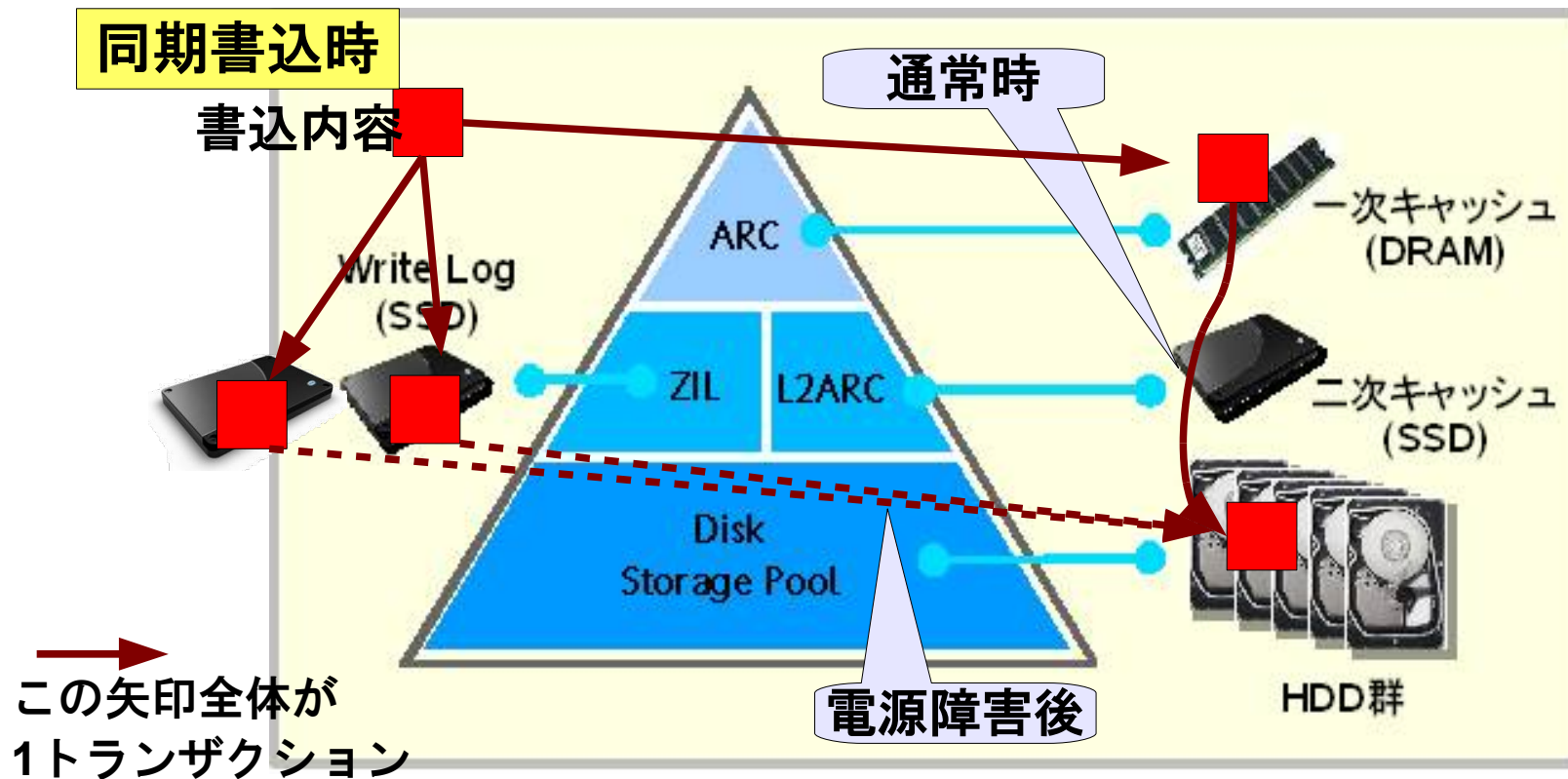
- 構成 A: Flash 無しのストレージプール (DRAM + 7x 10K RPM SAS)
- 構成 B: Flash 有りのストレージプール (DRAM + Read SSD + Log SSD + 5x 4200 RPM SATA)



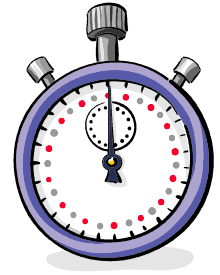
- Flash 利用で**コスト面も優位** 
- 大規模構成時のコストメリットが特に期待できる


# Flash (SSD)使用時の信頼性

- SSDの冗長構成が可能
- 読込SSD全故障: **単なるキャッシュミス扱い** 
- 書込SSD全故障: **一次キャッシュ中のコピー**  使用



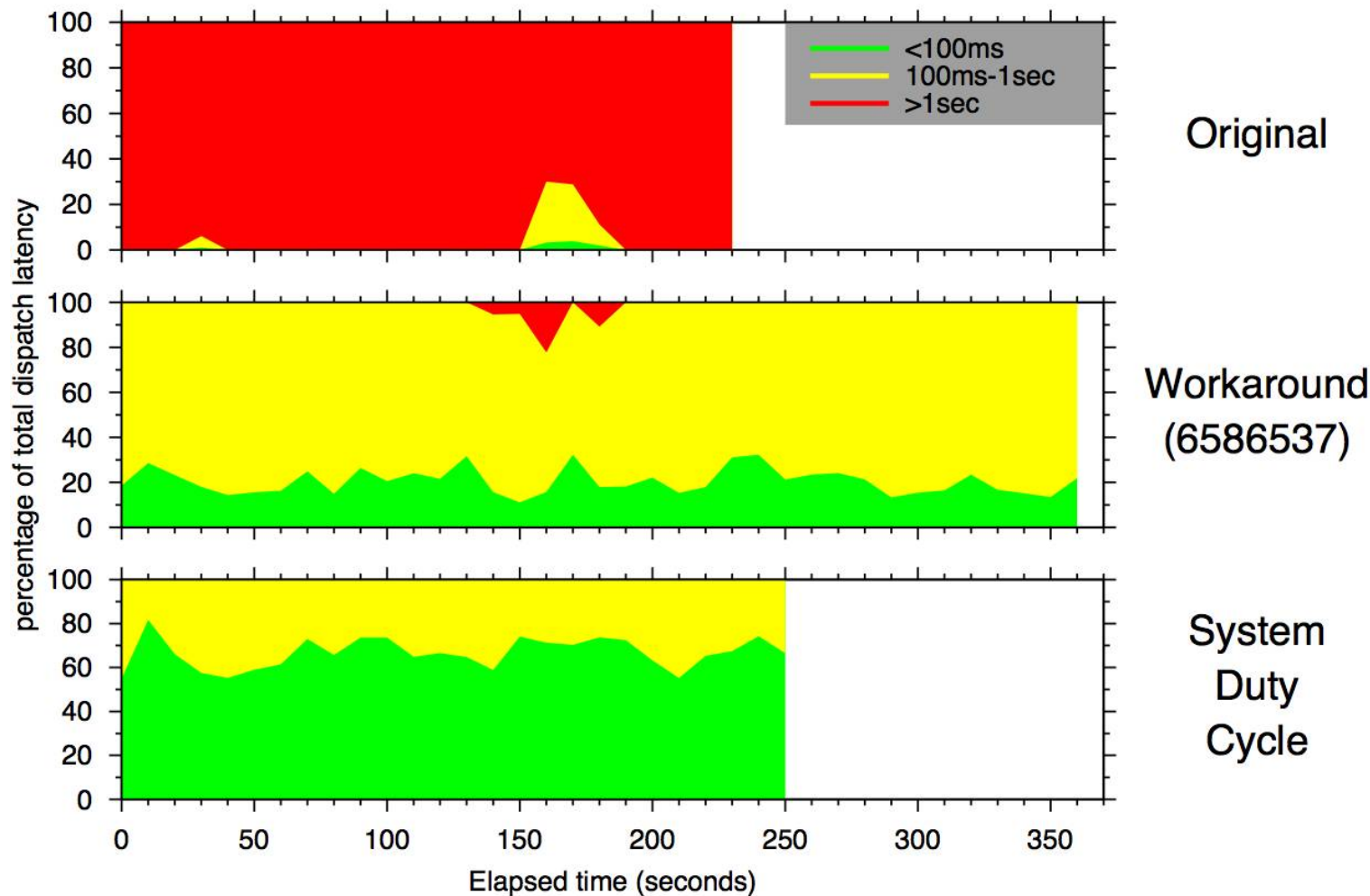
# duty-cycleスケジューラ



- これまでの問題点
  - ZFSトランザクション処理はCPU使用量が多い
    - トランザクションが多くなると、圧縮、暗号化、重複排除などと合わせて何秒か必要なことも
  - カーネルスレッドが高い優先度で動作
  - 結果: webサーバーやNFSなどの遅延が膨れ上がる
- 解決策 (Solaris 11 Express)
  - CPU使用量の大きいカーネルスレッド用スケジューリングクラス
  - 優先度を量子化・可変にしてオンオフを切替、他との調整
  - 結果: スループットへの影響極小で**遅延を大幅削減** 


# duty-cycleスケジューラの効果

Dispatch latency bubbles induced by ZFS IO threads

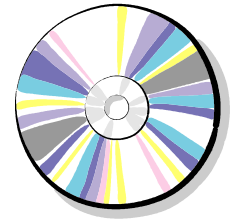



# logbias属性値



- これまでの問題点
  - slogデバイスは限りある資源
    - 遅延の許されないI/Oが多量の同期書込に埋もれてしまう
  - 全ての同期書込が低レイテンシである必要は無い
    - レイテンシが重要で無い場合はディスクに書く方が低コスト
    - ディスク本数が多ければ帯域もslogデバイスよりはるかに大
- 解決策 (Solaris 11 Express) 
  - データセットごとに**logbias属性値**を設定可能に
    - latency または throughput
  - 例: Oracle DBのredoログ: `zfs set logbias=latency`
  - 例: Oracle DBのデータファイル: `zfs set logbias=throughput`
  - 30%以上の性能向上

# sync属性値




- 同期書込の挙動をデータセットごとに制御
  - standard: 通常のPOSIXルールに従う (デフォルトかつ推奨)
    - fsync(), O\_DSYNC write() などの際、stableなストレージにコミットされてからreturn
  - always: 全トランザクションが同期に
    - 明示的な同期指定は不要
    - 通常は性能が低下するが、ハードウェアと負荷要件によっては性能向上することもある
  - disabled: 全てが非同期に
    - 高速slogデバイスが無い場面で大幅な性能向上
    - ただし、使用は**リスクを検討して慎重に** 
      - 書込が同期である必要が無い場面
      - DBやNFS操作など、アプリからの同期トランザクション要求は無視される
      - 使用中の / や /var への設定は仕様外動作、データロス、リプレイアタックの危険あり
  - sync属性値はzil\_disabledパラメータを置き換える

# RAID-Z/ミラー ハイブリッドアロケータ

- RAID-Zグループがミラーとしても使用される
  - RAID-Zとして使用: ユーザーデータと多くのメタデータ
    - 容量と書込スループットが最大化
  - ミラーとして使用: 低レイテンシが要求されるメタデータ
    - **読込IOPS最大化、読込レイテンシ最小化** !
    - **小さなランダムアクセスされるメタデータに有益**
      - データノード / 間接ブロック、ディレクトリ、重複排除テーブル
  - ミラープールのような耐障害性は提供せず、性能向上のみ
  - ミラー化メタデータ分の容量を消費する !
- 現実世界の負荷で**2-4倍の高速化**を達成したのものもある
  - 重複ブロックのある大きなファイルをコピー
  - ファイルの多いディレクトリでの `rm -rf`
- 既存プールはバージョンを29以降にアップグレードすれば以降の書込に適用される

## ZFSの性能向上

# カーネルモード iSCSIターゲット

- Solaris 10のiSCSI targetはユーザーランド実装 (shareiscsi で設定)
- Solaris 11 ExpressのCOMSTAR port providerは**カーネル内**実装
  - ZFSSA も同様 (COMSTAR 前と比べて 2.7倍の性能向上) 
  - WCE (Write Cache Enabled) 設定
    - HDDやHW-RAIDのWCEとより良く協調可能
  - **設定手順** (shareiscsiでは設定できない)
- 
  - stmfadm create-lu: backing store を logical unit provider に登録
  - itadm create-target: COMSTAR iSCSI port provider に紐付け



- iSCS MC/S (Multiple Connection per Session) も Solaris 11 Express 2010.11 に対応

# ルートプールに関する拡張点

# ルートプールに関する拡張点

- Live Upgradeの lu\* コマンド群は **beadm** コマンドに置き換え
- AI (Automated Installer) で以下の機能を提供
  - ミラー化ルートプール作成
  - スワップとダンプのサイズ変更
  - (LiveCDとテキストインストーラではこれらは行えない)
- pkg update / image-update コマンド
  - 新たな BE (ブート環境) を自動作成
  - パッチ適用やアップグレードの前に別途 BE (ブート環境) を作成する必要が無くなった
  - パッチを個別に適用する必要無し
- ミラーのattach時にブートブロックが自動適用
  - スペアへの切り替わり時も同様

## BEのアップグレード例

- 既存ZFS BEを pkg update で更新後に新BE zfsBE-1でブート

```
# pkg update ...
```

(...しばらく後、更新完了後)


```
A clone of zfsBE exists and has been updated and activated.  
On the next boot the Boot Environment zfsBE-1 will be mounted  
on '/'.
```

```
Reboot when ready to switch to this updated BE.
```

```
# init 6
```

(...しばらく後、リブート完了後)


```
# beadm list
```

BE	Active	Mountpoint	Space	Policy	Created
zfsBE	-	-	9.38M	static	2010-10-15 09:18
 <b>zfsBE-1</b>	NR	/	10.76G	static	2010-11-05 09:57

# プールの修復に関する拡張点

プールの修復に関する拡張点


## 書込順序を無視するディスクへの対応

- ・ プールの完全性は書込順序の明確性に依存 
  - ・ 安価なディスクやブリッジには**書込順序を無視するもの**もある
  - ・ 結果: 指し示すデータより先に uberblock が書き込まれたりする
  - ・ 突然の電源断などでプールへのアクセスができなくなる

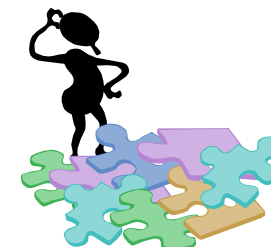
**ZFS**  
The Zettabyte File System



## 書込順序を無視するディスクへの対応策

- デバイスが書込順序を無視していても**プールを修復** 
  - プールのオープン時に最後のTXGが完全で一貫しているか確認
    - トランザクション作成時間を元にプールを高速スキャン
  - 完全性や一貫性が損なわれている場合
    - それより前の uberblockにロールバック
    - スキャンして確認して駄目ならロールバック、、を繰り返す
  - ブロック再利用を遅らせることでロールバックの信頼性向上
    - 直近で開放されたブロックは決してすぐには再利用しない
    - 内容が有効であることを仮定できるようにする

## ログが無い場合の修復



- ・ ログが無いプールをインポートしようとする、

```
# zpool import dozer
```

```
The device below are missing, use '-m' to import the pool anyway:
```

```
  c3t3d0 [log]
```

```
cannot import 'dozer': one or more devices is currently unavailable
```

- ・ インポート手順 (例は次ページ)
  - ・ **-m** オプション付きでインポート可能
- !
  - ・ 縮退モードでインポートされる
  - ・ インポート後にログデバイスを再attach可能
    - ・ zpool online使用
    - ・ ミラー化ログでも可能
  - ・ zpool clear でエラーを消去

## プールの修復に関する拡張点

# ログデバイスの無いプールのインポート例



```
# zpool import -m dozer
```

```
# zpool status dozer
```

```
pool: dozer
```

```
state: DEGRADED
```

```
status: One or more devices could not be opened. Sufficient replicas exist for the pool to continue functioning in a degraded state.
```

```
action: Attach the missing device and online it using 'zpool online'.
```

```
see: http://www.sun.com/msg/ZFS-8000-2Q
```

```
config:
```

NAME	STATE	READ	WRITE	CKSUM
dozer	DEGRADED	0	0	0
mirror-0	ONLINE	0	0	0
c3t1d0	ONLINE	0	0	0
c3t2d0	ONLINE	0	0	0
logs				
14685044587769991702	UNAVAIL	0	0	0

## プールの修復に関する拡張点

# 読込専用モードでのプールインポート

- プール損傷時にプール内のデータを救出できる可能性有

- 書込エラー発生時など

```
# zpool import -o readonly=on tank
```

```
# zpool scrub tank
```

```
cannot scrub tank: pool is read-only
```

- ファイルシステムとボリューム全てが読込専用でマウント
- そのプールでのトランザクション処理は不可となる
  - ZIL内で保留中の同期書き込みは保留のまま
    - プールが読み書き可能でインポートされるまで
- プールへの属性値変更は無視される
- エクスポートしインポートすることで読み書き可能に戻せる



## zpoolコマンドの機能拡張 (1)

- zpool statusが**より多くの** scrubやresilverの情報を出力
  - resilver 進行報告

```
scan: resilver in progress since Thu Oct 28 14:08:04 2010
      24.3M scanned out of 12.2G at 3.03M/s, 1h8m to go
      24.0M resilvered, 0.19% done
```

- resilver 完了メッセージ

```
scan: resilvered 12.2G in 0h14m with 0 errors on Thu Oct 28..
```

- scrub 進行報告

```
scan: scrub in progress since Thu Oct 28 13:50:22 2010
      98.6M scanned out of 13.3G at 16.4M/2, 0h13m to go
      0 repaired, 0.72% done
```


- scrub 完了メッセージ

```
scan: scrub repaired 512B in 1h2m with 0 errors on ...
```


- メッセージはシステム再起動後も**存続**

プールの修復に関する拡張点

## zpoolコマンドの機能拡張 (2)

- zpool list/status の新規オプション
-  インターバルと回数 の引数をとれる (\*statコマンド同様)

```
# zpool status [pool] [interval] [count]
```

- zpool list/status/iostat での  タイムスタンプ 表示
  - -T d は date(1)形式、-T u は time(2)形式

```
# zpool list -T d
2010年10月28日 (木) 15時04分07秒 JST
```

```
# zpool status -T u
1288300086
  pool: rpool
  state: ONLINE
  scan: non requested
config: .....
```

# zfs send/recvの拡張点

# zfs send/recvの拡張点

- send側とrecv側で異なる属性値を**send/recv時に設定**可能に
- send側の属性値を使用するよう指定することも可能
  - スナップショットを元にオリジナルを復元する際など
- recv側で属性値を無効にすることも可能



## zfs send/recvの拡張点

### 例 1

- tank/dataは圧縮属性が無効 (compression=off)
- recv側で圧縮属性を有効にする (compression=on)

```
# zfs get compression tank/data
```

NAME	PROPERTY	VALUE	SOURCE
tank/data	compression	off	default

```
# zfs send -p tank/data@snap1 | zfs recv -o compression=on \  
-d bpool
```

```
# zfs get -o all compression bpool/data
```

NAME	PROPERTY	VALUE	RECEIVED	SOURCE
bpool/data	compression	on	off	local

### 例 2

- tank/dataをバックアップ用プールにsend
  - オリジナルtank/dataの復元用
- オリジナルの圧縮属性値を zfs send -b で保持

```
# zfs get compression tank/data
```

NAME	PROPERTY	VALUE	SOURCE
tank/data	compression	off	default

```
# zfs send -b tank/data@snap1 | zfs recv -d restorepool
```

```
# zfs get -o all compression restorepool/data
```




NAME	PROPERTY	VALUE	RECEIVED	SOURCE
restorepool/data	compression	off	off	received

### 例 3

- send側属性値を zfs recv -x で無効化
- ホームディレクトリを再帰的にsendし、recv側ではquota属性値だけ無効化する例



```
# zfs send -R tank/home@1020 | zfs recv -x  quota bpool/home
```

```
# zfs get -r quota bpool/home
```

NAME	PROPERTY	VALUE	SOURCE
bpool/home	quota	 <b>none</b>	default
bpool/home@1020	quota	-	-
bpool/home/cindys	quota	<b>none</b>	local
bpool/home/cindys@1020	quota	 -	-
bpool/home/tom	quota	<b>none</b>	local
bpool/home/tom@1020	quota	 -	-


# その他の新機能、拡張点

# ZFS その他の新機能、拡張点

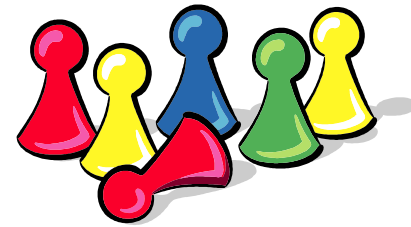
- ユーザー/グループ単位の容量制限 (user/group quota)
- トリプルパリティ RAID-Z
- ZFS スナップショット差分 (zfs diff)
- shadow migration
- zpool split
  - ミラー化プールを2個の同一プールに**分割** 
  - 用途
    - ディザスタリカバリ
    - サイトの複製
    - 物理的アーカイブを簡単に取得可能
- zfs send/recv による NDMP バックアップ
  - send/recv の速度で 
  - 既存 NDMP 環境に**容易に統合**可能
- 動的 LUN 拡張 (autoexpand属性値)



# ユーザー・グループ単位の容量制限 (user/group quota)

- エンタープライズ向けとして
  - 「自分用の残り容量はどこへ？」という場面への対応
- 教育機関向けとして
  - 細かく制御できない多くのユーザー
    - ユーザーごとの容量制限をかけたい
  - **ユーザーごとのファイルシステム提供は効率が悪い場面に対応** 
- 容量制限のされ方
  - 制限値を1トランザクションで越えることもできる
  - 制限値を越えたら、制限値を下回るまでEDQUOTが返る
- SMBのSIDとPOSIXのUID/GIDをサポート

## 属性値



- データセット属性
  - `userused@<user>`
  - `userquota@<user>`
  - `groupused@<group>`
  - `groupquota@<group>`
- `<user>`や`<group>`の種類は以下
  - POSIX ID の数値 (例: 77196)
  - POSIX のユーザー名 (例: maybee)
  - SID の数値 (例: S-1-123-456-789)
  - SID のユーザー名 (mark.maybee@oracle)
- 他の属性値同様 `zfs get/set` を使用

# サブコマンド

- zfs userspace / groupspace
  - 1行ずつユーザーまたはグループの容量制限情報を表示


TYPE		NAME	USED	QUOTA
POSIX User		ahrens	14M	1G
POSIX User		lling	258M	none
POSIX Group		staff	3.75G	32T
SMB User		marks@oracle	103M	5G

# トリプルパリティRAID-Z (RAID-Z3)

- ディスク3本の障害に耐性 (業界初)
  - ディスク2本障害後のresilver時にリードエラー発生時も同様
    - ドライブ容量増はresilver時間増を伴いこうした可能性も増加
    - ! **ダブルパリティを越える冗長性**が求められる
- より大容量、高速、「低」信頼ドライブを使用可能に
  - 昨今のHDDは容量の30-40%がECC
  - ディスク管理情報書き換えで以下のようなことが実現できる可能性
    - ドライブあたり30-40%使用可能量増加
    - ドライブあたり30-40%の帯域
    - ZFSがより多くのI/Oエラーを検知・修正

18 HDD	1 group	可用性考慮
RAID-Z	17D+1P	6x (2D+P)
RAID-Z2	16D+2P	2x (7D+2P)
RAID-Z3	15D+3P	1x (15D+3P)

# ZFS スナップショット差分 (zfs diff)(1)

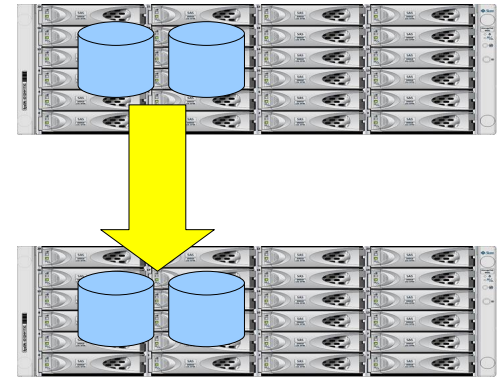
- スナップショットの差分を**確認可能** 
  - ファイルの追加、削除、修正、名前変更が表示される



```
# zfs create tank/jeff
# cd /tank/jeff
# echo hello > hello.txt
# echo world > world.txt
# zfs snapshot tank/jeff@hello-world
# echo everyone > everyone.txt
# mv hello.txt goodbye.txt
# rm world.txt
# ls
goodbye.txt    everyone.txt
# zfs diff tank/home/jeff@hello-world
M      /tank/jeff/
R      /tank/jeff/hello.txt    -> /tank/jeff/goodbye.txt
-      /tank/jeff/world.txt
+      /tank/jeff/everyone.txt
```

# ZFS スナップショット差分 (zfs diff)(2)

- スナップショットの差分表記
  - M: ファイルまたはディレクトリの変更、ディレクトリのリンク数変更
  - -: ファイルまたはディレクトリが、古いスナップショットには存在し、新しいスナップショットには存在しない
  - +: ファイルまたはディレクトリが、新しいスナップショットには存在し、古いスナップショットには存在しない
  - R: ファイルまたはディレクトリの名前が変更された

# shadow migration



- ZFS ストレージアプライアンスの機能
  - 1年以上前から利用可能な機能
- ファイルシステムをZFSに移行
  - 既存ファイルシステムを読込専用でマウント
  - 新規ファイルシステムは既存ファイルシステム同様にファイル提供
  - 既存ファイルシステムのファイルやディレクトリが**即座にアクセス可** 
    - ファイルシステムの完全移行(コピー完了)を待つ必要が無い
    - 名前変更、ファイル更新なども可能
  - 新規ファイルシステムはネットワーク**共有も可能** 
  - ユーザーアクセスに対応する裏で移行作業用スレッドが動作

# まとめと参考情報



# まとめ



- 変わらないもの
  - ZFS: 革新的ファイルシステム
    - ファイルシステムとして重要なポイントを網羅
      - 拡張性、堅牢性、機能、性能、管理性
  - ZFS: 広範なインフラ向けソリューションでの理想的なストレージ基盤
    - 多くの機能を低コストで
    - 多くの応用例、多くの事例
    - クラウド時代に最適な拡張性とストレージ仮想化
- **Solaris の強化・進化を支え、Solaris とともに強化・進化**
  - ZFS のフル活用で Solaris も強力に
  - 時代の要請に**応えつつ次世代**エンタープライズを見据える



# Solaris 11 Express の ZFS 参考情報

- Oracle Solaris 11 Express - ZFS Administration Guide
  - <http://download.oracle.com/docs/cd/E19963-01/821-1448/>
  - <http://download.oracle.com/docs/cd/E19963-01/821-1448/821-1448.pdf>
- ZFS Best Practice / Troubleshooting Guide
  - <http://www.solarisinternals.com>
- ZFS Dedup and Encryption FAQ:
  - <http://hub.opensolaris.org/bin/view/Community+Group+zfs/faq>
- zfs-discussion
  - <http://opensolaris.org/jive/forum.jspa?forumID=80>

# ZFS 参考情報

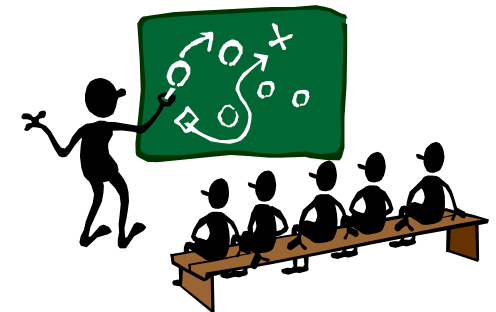
## 参考情報リンク集

hiroa blog zfs

検索

からリンクしました!

- 解説記事、アーキテクチャ、ソースツアー、デモ、マニュアル、構築時の参考情報、blog、事例、書籍
- Oracle University コース
  - Solaris 10 ZFS管理 (SA-2290-S10)
  - ZFSの機能、プールとファイルシステム管理、スナップショットとクローンの操作、ACL、トラブルシューティング
- OTN (Oracle Technology Network)
  - <http://www.oracle.com/technetwork/jp/index.html> (Japan)
  - <http://www.oracle.com/technetwork/index.html> (Global)
  - 過去の ZFS ダイセミ資料も **OTNオンデマンドで公開** 



# ZFS 参考資料 (1)

- BigAdmin System Administration Portal  
ZFS - Sunの最新ファイルシステム
  - ストレージの完全性、安全性、  
およびスケーラビティー
    - [http://www.sun.com/bigadmin/hubs/multilingual/japanese/content/zfs\\_part1.scalable.jsp](http://www.sun.com/bigadmin/hubs/multilingual/japanese/content/zfs_part1.scalable.jsp)
  - 管理の簡素化と将来の拡張機能
    - [http://www.sun.com/bigadmin/hubs/multilingual/japanese/content/zfs\\_part2\\_ease.jsp](http://www.sun.com/bigadmin/hubs/multilingual/japanese/content/zfs_part2_ease.jsp)
- OpenSolaris Community: ZFS
  - <http://opensolaris.org/os/community/zfs>
- ZFSソースツアー
  - <http://www.sun.com/bigadmin/hubs/multilingual/japanese/content/zfs-source.jsp>
- 明日からでも使いたい次世代ファイルシステムノート  
PCでこそ使いたいZFS
  - <http://www.atmarkit.co.jp/news/200706/29/zfs.html>
- Solaris ZFS集中講座
  - <http://www.atmarkit.co.jp/fserver/index/zfs.html>
- OpenSolaris でサーバ構築第4回  
一瞬でのバックアップを実現する  
Solaris ZFS
  - <http://www.atmarkit.co.jp/flinux/rensai/opensolaris04/opensolaris04a.html>
- ZFS デモ
  - <http://www.atmarkit.co.jp/news/200710/09/solaris.html>
  - <http://jp.youtube.com/watch?v=N8eu06HNj1w>
  - <http://jp.youtube.com/watch?v=1zw8V8g5eT0>

hiroa blog zfs

検索

からリンクしました!

# ZFS 参考資料 (2)

からリンクしました!

## 事例

- <http://wikitech-static.wikimedia.org/articles/z/f/s/Zfs.html>
- <http://www.oracle.com/us/corporate/customers/060399.pdf>
- [http://hepix.caspur.it/storage/hep\\_pdf/2007/Spring/zfsatdesy.pdf](http://hepix.caspur.it/storage/hep_pdf/2007/Spring/zfsatdesy.pdf)
- [http://blogs.sun.com/video/entry/ourstage\\_interview](http://blogs.sun.com/video/entry/ourstage_interview)
- <http://www.oracle.com/us/corporate/customers/060413.pdf>
- <http://sun.systemnews.com/articles/133/1/OpenStorage/21332>
- [http://www1.jp.dell.com/content/topics/segtopic.aspx/misc/seminars/OSS\\_seminar0901?c=jp&l=ja&s=gen](http://www1.jp.dell.com/content/topics/segtopic.aspx/misc/seminars/OSS_seminar0901?c=jp&l=ja&s=gen)
- <http://www.oracle.com/us/corporate/customers/customersearch/index.html?Keyword=zfs>

## 構築時の参考情報

- [http://www.solarisinternals.com/wiki/index.php/ZFS\\_for\\_Databases](http://www.solarisinternals.com/wiki/index.php/ZFS_for_Databases)
- [http://www.solarisinternals.com/wiki/index.php/ZFS\\_Best\\_Practices\\_Guide](http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide)
- <http://wikis.sun.com/display/BluePrints/Configuring+Sun+Storage+7000+Systems+for+Oracle+Databases>
- <http://wikis.sun.com/display/BluePrints/Best+Practices+for+Running+Oracle+Databases+in+Solaris+Containers>

## 書籍『ZFS 仮想化されたファイルシステムの徹底活用』

# **Hardware and Software** **Engineered to Work Together**

**ORACLE®**

# 付録: 追加情報



# 参考情報: スワップ領域の暗号化

- ZVOL でのスワップ領域作成時に
  - `zfs set primarycache=metadata rpool/swap`
  - `zfs set secondarycache=none rpool/swap`
- `/etc/vfstab`内のエントリ
  - ラップ用鍵を `/dev/random` で生成
  - 暗号化スワップ領域は再起動で削除され再作成される
  - ZVOL では無い場合は暗号化 `lofi` が使用される

```
#device          device  mount FS      fsck mount  mount
#to mount        to fsck point type  pass at boot options

/dev/zvol/dsk/rpool/swap -      -      swap -      no      encrypted
```

## 参考情報: ダンプ用ZVOLの暗号化

- ダンプ用ZVOLは事前割当済み
  - I/OはZIOのパイプラインを経由しない
- ダンプ用ZVOLも暗号化は可能
- AES CTR モードを使用する (CCM/GCMでなく)
  - ダンプはクラッシュシステムの事後デバッグ用途なので、データの長期保存やそこでのプログラム実行は行わない
  - こうした用途としては十分な秘匿性が確保される
- ラップ用鍵が無いと fmd サービスが maintenance状態に
  - `svc:/system/fmd` は `svc:/system/dumpadm`に依存するので

# 参考情報: ZFS ACL相互運用性向上 (1)

- 通常のACLの場合はdeny エントリが不要に
  - 通常: 0644, 0755, 0664
  - 0705, 0060 などは deny エントリが必要
- 以前の表示例

```
# ls -v file.1
-rw-r--r--    1 root    root    206663 Jun 14 11:52 file.1
0:owner@:execute:deny
1:owner@:read_data/write_data/append_data/write_xattr/
write_attributes/write_acl/write_owner:allow
2:group@:write_data/append_data/execute:deny
3:group@:read_data:allow
4:everyone@:write_data/append_data/write_xattr/execute/
write_attributes/write_acl/write_owner:deny
5:everyone@:read_data/read_xattr/read_attributes/read_acl/
synchronize:allow
```

# 参考情報: ZFS ACL相互運用性向上 (2)

- Solaris 11 Express 2010.11の表示例
  - モードが0644の場合はdenyエントリを含まない

```
# ls -v file.1
-rw-r--r--    1 root    root    206663 Jun 14 11:54 file.1
0:owner@:read_data/write_data/append_data/write_xattr/
write_xattr/read_attributes/write_attributes/read_acl/
write_acl/write_owner/synchronize:allow
1:group@:read_data/read_xattr/read_attributes/read_acl/
synchronize:allow
2:everyone@:read_data/read_xattr/read_attributes/read_acl/
synchronize:allow
```

## 参考情報: ZFS ACL相互運用性向上 (3)

- 元の変更されていないパーミッションを保持するため継承の際 ACL は複数のACEに分割されない。  
パーミッションはファイル作成時のモードを強制するため必要に応じて修正される
- `aclinherit=restricted`のとき削減されたパーミッション情報を含み、継承の際にACLが複数のACEに分割されないことを意味する
- デフォルトでは`chmod(2)`でACLは削除される。この変更は`aclmode`属性値が無くなったことも意味する。

## 参考情報: ZFS ACL相互運用性向上 (4)

- 新たなパーミッションモード計算ルールは、ACLがファイルオーナーのユーザーACEを持つなら、そのパーミッションはパーミッションモードの計算に含まれる。グループについても、ACEがファイルのグループ所有者のときに同じルールが適用される